US-PAT-NO: 6477580

DOCUMENT-IDENTIFIER: US 6477580 B1

TITLE: Self-described stream in a communication services patterns environment

---------- KWIC ---------

Use report writers when users require easy and quick access to corporate data.
Since developers can deliver reports as run-time applications, users are
shielded from having to learn complicated databases in order to access
information. All a **user has to do to retrieve the data** is click on an icon to
launch a report. Because these run-time applications are smaller than normal
applications, they launch faster and require very little training to operate.
(source is market research)

Features to look for include but are not limited to:
WYSIWYG print preview
Ability to create views--prevents users from getting overwhelmed with choices
when selecting a table, acts as a security system by controlling which **users**
have access to certain data, and increases performance since only the **data**
users need gets **downloaded** to the report engine, thereby reducing network
traffic. Data dictionary--store predefined views, formats, and table and field
name aliases User friendly query tool Scripting or macro language Supported
data types and formats Formatting capabilities (page orientation, fonts,
colors, margins, condensed printing, etc.) Supported report types Aggregate
functions.

Database Services are responsible for providing access to a local or a remote database, maintaining integrity of the data within the database and supporting the ability to store data on either a single physical platform, or in some cases across multiple platforms. These services are typically provided by DBMS vendors and accessed via embedded or call-level SQL variants and supersets. Depending upon the underlying storage model, non-SQL access methods may be used instead.

Security components can restrict access to functions within an application based on a users security level. The highest level security is whether the user has access to run the application. The next level checks if the user has access to functions within the application, such as service calls or windows. At an even lower level, the security component could check security on more granular functions, such as widgets on a window.

Communications Security services control access to network-attached resources. Combining network Security services with security services in other parts of the system architecture (e.g., application and database layers) results in robust security.

Applications may compose and transfer faxes as part of notifying users or delivering information. For example, an application may use Fax services to add customer-specific information to a delivery receipt form and fax the form to a customer.

Terminal services allow a client to connect to a non-local host via a network and to emulate the profile (e.g., the keyboard and screen characteristics)

required by the host application. For example, when a workstation application
logs on to a mainframe, the workstation functions as a dumb terminal. Terminal
Services **receive user input and send data** streams back to the host processor.
If connecting from a PC to another PC, the workstation might act as a remote
control terminal (e.g., PCAnywhere).

What **level of security** is required?

E-Mail takes on a greater significance in the modem organization. The E-Mail
system, providing it has sufficient integrity and stability, can function as a
key channel through which work objects move within, and between organizations
in the form of messages and electronic forms. An E-Mail server stores and
forwards E-Mail messages. Although some products like Lotus Notes use
proprietary protocols, the following protocols used by E-Mail Services are
based on open standards: X.400--The X.400 message handling system standard
defines a platform independent standard for store-and-forward message transfers
among mail servers. X.400 is often used as a backbone e-mail service, with
gateways providing interconnection with end-user systems. SMTP--Simple Mail
Transfer Protocol (SMTP) is a UNIX/Internet standard for transferring e-mail
among servers. MIME--Multi-Purpose Internet Mail Extensions (MIME) is a
protocol that enables Internet users to exchange multimedia e-mail messages.
POP3--Post Office Protocol (POP) is used to distribute e-mail from an SMTP
server to the actual recipient. IMAP4--Internet Message Access Protocol,
Version 4 (IMAP4) allows a client to access and manipulate electronic mail
messages on a server. IMAP4 permits manipulation of remote message folders,
called "mailboxes", in a way that is functionally

equivalent to local
mailboxes. IMAP4 also provides the capability for an off-line client to
re-synchronize with the server. IMAP4 includes standards for message handling
features that allow **users to download message header information** and then
decide which e-mail message contents to **download**.

Communications Security services control access to network-attached resources.
Combining network Security services with security services in other parts of
the system architecture (e.g., application and database **layers) results in
robust security**.

There are also varying methods of employing encryption types described above to
encrypt data sent across a network: Data link layer--data is encrypted before
it is placed on the wire. Data link encryptors are generally hardware
products. Application layer--data is encrypted by the application. Netscape's
Secure Sockets Layer (SSL) is one example of application-layer encryption for
WWW browsers. SSL uses RSA encryption to wrap security information around
TCP/IP based protocols. Network layer--data is encrypted inside the network
layer header, therefore relying on the network layer protocol.

Netscape's **Secure Sockets Layer** (SSL); S-HTTP; e-mail encryption; S-MIME

Encryption that is architected into Web-based solutions Netscape's **Secure
Sockets Layer** (SSL)--provides encryption for World Wide Web browsers.
S-HTTP--a secure version of the HTTP data transfer standard; used in
conjunction with the World Wide Web.

The following are examples of ways to implement Authorization services: Network

Operating Systems--Authorization services are bundled with all network operating systems in order to control user access to network resources. Firewall Services protect sensitive resources and information attached to an Intxxnet network from unauthorized access by enforcing an access control policy. A variety of mechanisms exist for protecting private networks including: Filters--World Wide Web filters can prevent users from accessing specified content or Internet addresses. Products can limit access based on keywords, network addresses, time-of-day, user categories, etc. Application Proxies--An application-level proxy, or application-level gateway, is a robust type of firewall. (A firewall is a system that enforces an access control policy between a trusted internal network and an untrusted external network.) The application proxy acts at the application level, rather than the network level. The proxy acts as a go-between for the end-user by completing the user-requested tasks on its own and then transferring the information to the user. The proxy manages a database of allowed user actions, which it checks prior to performing the request. Servers, Applications, and Databases--Authorization can occur locally on a server to limit access to specific system resources or files. Applications and databases can also authorize users for specific levels of access within their control. (This functionality is within the Environment Services grouping in the execution architecture.)

Authentication can occur through various means: Basic Authentication--requires that the Web client supply a user name and password before servicing a request. Basic Authentication does not encrypt the password in any

way, and thus the
password travels in the clear over the network where it could be detected with
a network sniffer program or device. Basic authentication is not secure enough
for banking applications or anywhere where there may be a financial incentive
for someone to steal someone's account information. Basic authentication is
however the easiest mechanism to setup and administer and requires no special
software at the Web client. ID/Password Encryption--offers a somewhat higher
level of security by requiring that the user name and password be encrypted
during transit. The user name and password are transmitted as a scrambled
message as part of each request because there is no persistent connection open
between the Web client and the Web **server**. Digital Certificates or
Signatures--encrypted digital keys that are issued by a third party "trusted"
organization (i.e. Verisign); used to verify user's authenticity. Hardware
tokens--small physical devices that may generate a one-time password or that
may be inserted into a card reader for authentication purposes. Virtual
tokens--typically a file on a floppy or hard drive used for authentication
(e.g. Lotus Notes ID file). Biometric identification--the analysis of
biological **characteristics to verify individuals** identify (e.g., fingerprints,
voice recognition, retinal scans).

An intelligent communications fabric monitors and routes data flows and
provides functionality (security, directories, etc.) to clients and servers.
An intelligent communications fabric provides the following benefits: An
intelligent network can manage itself, including addressing, routing, security,
recovery, etc. It is inefficient for individual clients and servers to perform

such tasks. Specialized network components reduce the network-related processing that occurs on clients and servers. An intelligent network integrates heterogeneous clients, servers, and other resources by resolving incompatible protocols and standards. An intelligent network has the capability to actively manage the flow of information rather than simply moving data. This allows the network to effectively deliver multimedia and other network-sensitive traffic. An intelligent network adds value to enterprise resources by presenting a cohesive view of available resources and increasing the **level of security** associated with those resources.

The Packet Forwarding/Internetworking service transfers data packets and manages the path that data takes through the network. It includes the following functionality: Fragmentation/Reassembly--The Packet Forwarding/Internetworking service divides an application message into multiple packets of a size suitable for network transmission. The **individual packets include information** to allow the **receiving** node to reassemble them into the message. The service also validates the integrity of received packets and buffers, reorders, and reassembles packets into a complete message. Addressing--The Packet Forwarding/Internetworking service encapsulates packets with addressing information. Routing--The Packet Forwarding/Internetworking service can maintain routing information (a view of the network topology) that is used to determine the best route for each packet. Routing decisions are made based on the cost, percent utilization, delay, reliability, and similar factors for each possible route through the network. Switching--Switching is the process of receiving a packet, selecting an appropriate

outgoing path, and sending the packet. Switching is performed by routers and switches within the communications fabric. Switching can be implemented in the following ways: For some network protocols (e.g., IP), routers draw upon dynamic routing information to switch packets to the appropriate path. This capability is especially important when connecting independent networks or subnets. For other network protocols (e.g., Ethernet, Token Ring), switching simply directs packets according to a table of physical addresses. The switch can build the table by "listening" to network traffic and determining which network nodes are connected to which switch port. Some protocols such as Frame Relay involve defining permanent routes (permanent virtual circuits PVCs) within the network. Since Frame Relay is switched based upon PVCs, routing functionality is not required. Multicasting--The Packet Forwarding/Internetworking service may support multicasting, which is the process of transferring a single message to multiple recipients at the same time. Multicasting allows a sender to transfer a single copy of the message to the communications fabric, which then distributes the message to multiple recipients.

Media **Access services manage the low-level** transfer of data between network nodes. Media Access services perform the following functions: Physical Addressing--The Media Access service encapsulates packets with physical address information used by the data link protocol (e.g., Ethernet, Frame Relay). Packet Transfer--The Media Access service uses the data link communications protocol to frame packets and transfer them to another computer on the same network/subnetwork. Shared Access--The Media Access service provides a method

for multiple network nodes to share access to a physical network. Shared Access schemes include the following: CSMA/CD--Carrier Sense Multiple Access with Collision Detection. A method by which multiple nodes can access a shared physical media by "listening" until no other transmissions are detected and then transmitting and checking to see if simultaneous transmission occurred. token passing--A method of managing access to a shared physical media by circulating a token (a special control message) among nodes to designate which node has the right to transmit. multiplexing--A method of sharing physical media among nodes by consolidating multiple, independent channels into a single circuit. The independent channels (assigned to nodes, applications, or voice calls) can be combined in the following ways: time division multiplexing (TDM)--use of a circuit is divided into a series of time slots, and each independent channel is assigned its own periodic slot. frequency division multiplexing (FDM)--each independent channel is assigned its own frequency range, allowing all channels to be carried simultaneously. Flow Control--The Media Access service manages the flow of data to account for differing data transfer rates between devices. For example, flow control would have to limit outbound traffic if a receiving machine or intermediate node operates at a slower data rate, possibly due to the use of different network technologies and topologies or due to excess network traffic at a node. Error Recovery--The Media Access service performs error recovery, which is the capability to detect and possibly resolve data corruption that occurs during transmission. Error recovery involves the use of checksums, parity bits, etc. Encryption--The Media Access service may perform encryption. (Note that

encryption can also occur
within the Communications Services layer or the Transport Services layer.)
Within the Network Media Services layer, encryption occurs as part of the data
link protocol (e.g. Ethernet, frame relay). In this case, all data is
encrypted before it is placed on the wire. Such encryption tools are generally
hardware products. Encryption at this level has the advantage of being
transparent to higher level services. However, because it is dependent on the
data link protocol, it has the disadvantage of requiring a different solution
for each data link protocol.

Specialized services convert between addresses at the Media **Access level** (i.e.,
physical addresses like Ethernet) and the Packet Forwarding/Internetworking
level (i.e., network addresses like IP). The following protocols are examples
of this functionality: ARP (Address Resolution Protocol)--allows a node to
obtain the physical address for another node when only the IP address is known.
RARP (Reverse Address Resolution Protocol)--allows a node to obtain the IP
address for another node when only the physical address is known.

Besides system **level security** such as logging into the network, there are
additional security services associated with specific applications. These
include: User Access Services--set of common functions that limit application
access to specific users within a company or external customers. Data Access
Services--set of common functions that limit access to specific data within an
application to specific users or user types (e.g., secretary, manager).
Function Access Services--set of common functions that limit access to specific
functions within an application to specific users or user

types (e.g.,
secretary, manager).

Object Request Broker (ORB) services, based on COM/DCOM and CORBA, focus on how components communicate. Component Framework Services, also based on CORBA and COM/DCOM, focus on how components should be built. The currently 2 dominant Component Frameworks include: 1. ActiveX/OLE--ActiveX and Object Linking and Embedding (OLE) are implementations of COM/DCOM. ActiveX is a collection of facilities forming a framework for components to work together and interact. ActiveX divides the world into two kinds of components: controls and containers. Controls are relatively independent components that present well defined interfaces or methods that containers and other components can call. Containers implement the part of the ActiveX protocol that allows for them to host and interact with components--forming a kind of back plane for controls to be plugged into. ActiveX is a scaled-down version of OLE for the Internet. OLE provides a framework to build applications from component modules and defines the way in which applications interact using **data transfer, drag-and-drop** and scripting. OLE is a set of common services that allow components to collaborate intelligently. In creating ActiveX from OLE 2.0, Microsoft enhanced the framework to address some of the special needs of Web style computing. Microsofts Web browser, Internet Explorer, is an ActiveX container. Therefore, any ActiveX control can be downloaded to, and plugged into the browser. This allows for executable components to be interleaved with HTML content and downloaded as needed by the Web browser. 2. JavaBeans--is Sun Microsystems proposed framework for building Java components and

containers. The intent is to develop an API standard that will allow components developed in Java (or beans), to be embedded in competing container frameworks including ActiveX or OpenDoc. The JavaBeans API will make it easier to create reusable components in the Java language.

Functional Criteria 1. Report Repository: The report architecture should work with, and support maintenance of, a report repository on the platforms within the client/server architecture. The report repository contains the detailed definitions of the reports. 2. Workgroup Report Support: The report architecture should work with and support distribution of reports generated on the workgroup **server**. 3. On-Demand Reports: The report architecture must support distribution of reports **requested** by users on demand. Typically, these reports will not have a set schedule or frequcency for distribution. The report architecture must support distribution of these reports without the requirement of manual or user intervention (subsequent to initial set up and conversion). 4. Scheduled Reports: The report architecture must support distribution of regularly scheduled reports. Typically, these reports will have a set schedule and frequency for distribution. The report distribution package must support distribution of these reports without the requirement of manual or user intervention (subsequent to set up and conversion). 5. Online Preview: The report architecture should allow preview of reports online from a user's intelligent workstation prior to actual distribution. Ideally, the report architecture itself would provide support for online preview of reports through software located on the intelligent workstation. 6. Graphical User Interface: The architecture should provide users with a

graphical user interface. 7. Bilingual Support: For companies where two or more languages are used, the report architecture must provide a multi-national user interface. (Note that large report runs targeted for multiple users may require the ability to change languages during the report.) 8. Basic Preview Functions: The report architecture should support basic preview functions. These include: Scrolling up and down. Scrolling left and right. Advancing to end or beginning of report without scrolling through intermediate pages. 9. Advanced Preview Functions: In addition to the basic preview functions listed previously, certain advanced preview functions may also be necessary: Page indexing (allows users to jump to specific report pages). Section indexing (allows users to jump to specific report sections). Search capabilities (allows users to search report for occurrence of a specific data stream). 10. Report **Level Security:** Reports may occasionally contain sensitive information. It is therefore important that access to certain reports be restricted to authorized users. The report architecture should provide a mechanism for implementing report **level security**. This security must be in place on all platforms with the client/server architecture. At the workgroup **level, the security** may consist of downloading sensitive report files to a secure directory, and having the LAN **administrator** release the report as appropriate. 11. Section, Page, and Field **Level Security:** Defining security at the report section, page, or field level would provide greater flexibility in determining and implementing report security. This is a desirable, though not mandatory, requirement of the report architecture. 12. Background Processing: The report

architecture should support the processing of reports in the background while the application works in the foreground during online hours. In other words, processing of reports should not negatively affect online response times, or tie up the user's workstation. 13. Automatic Report Addressing: The report architecture should provide a "humanly intelligible" address for all distributed reports. The address may be used by a print site operator, LAN **administrator,** or other personnel to manually sort printed output (if required). This criterion can be satisfied by automatic creation of banner pages or other means. 14. Delivery Costing: To provide sufficient **information to users** to avoid accidentally **downloading** or printing very large reports during peak usage hours, a distribution costing function can be useful. This function would warn users of reports that would overload the network or a printer. This costing function might provide recipient with a rough estimate of the amount of time that distribution might take. Finally, during the online day, the delivery costing mechanism might disallow transmission of reports that exceed a predetermined cost. 15. Multiple Destinations: The report architecture should support distribution of a single report to single or multiple destinations. 16. Destination Rationalization: For some systems, it is possible that multiple copies of a report will be sent to the same site--to several different users, for example. In these cases, it is highly desirable to have the report architecture recognize these situations whenever possible and distribute the specified report only once. 17. Automatic Printing: The report architecture should provide automatic print capabilities. Once a report has been distributed for printing (either through a "push"

distribution
scheduling mechanism or through a "pull" user **request**) no further user or
operations personnel involvement should be necessary to print the report at the
specified location. 18. Multiple Print Destinations: The report architecture
should support distribution of reports for printing at centralized, remote, or
local print sites without user or operations personnel intervention. 19.
Variable Printer Types: Printing on multiple types of printers, including line,
impact, and laser printers, should be supported. This should not require user
intervention--that is, the user should not have to specify the type of target
printer. Ideally, the report architecture would default this **information from
the user's** profile or the default printer defined in the local operating
system. This criterion requires that the report architecture support several
print mechanisms, such as postscript drivers and host/mainframe protocols (for
example, Advanced Function Printing [AFP]). 20. Variable Printer
Destinations: The report architecture should default the destination printer
for a specific report (from the user's profile or operating system parameters).
Additionally, the architecture should allow the user to change the printer
specified. Validation of the print destination also should be included. 21.
Special Forms Printing: The report architecture should support distribution of
"regular" reports and special forms reports. 22. Font Support: Some reports
may be printed on laser printers and/or may support electronic forms text
(i.e., including the forms text in the report dataset as opposed to printing
the report dataset on a pre-printed form). The architecture should allow
multiple fonts to be specified. 23. Report Archival: The report architecture

should provide and/or facilitate archival or disposition of report datasets.
Ideally, the architecture would permit definition of retention periods and
disposition requirements. 24. Report **Download:** The report architecture should
allow distribution of the **information contained in a report dataset to a user's**
intelligent workstation. The information should be in a form that can be
imported to a local word processing software, decision support software
package, or other appropriate application. 25. Application Transparency: It
is desirable for the report architecture to appear to the users as if it were
part of the overall application. This does not necessarily mean that the
architecture must integrate seamlessly with the application; a message
interface between the systems might be acceptable. 26. Selective Printing: It
would be desirable for the report architecture to provide users with the
ability to print only selected pages or sections of the report. This should
reduce paper usage, while still allowing **users to obtain a hard copy of the**
information as required. 27. Print Job Restart: It would be desirable if the
report architecture allowed a print job to be restarted from the point of
failure rather than having to reprint the entire report. This of particular
concern for very large reports.

Issues to consider include the following: (1) samples and assists that are
available to the developer; (2) existence of a scripting or programming
language; (3) **granularity of the security, or in other words, at what levels**
can security be added; (4) freedom of choosing productivity applications; (5)
existence of aggregate functions which allow for analysis of the workflow
efficiency; (6) existence/need for Business Processing

Re-engineering tools.

A pattern is "an idea that has been useful in one practical context and will probably be useful in others." Think of them as blueprints, or designs for proven solutions to known problems. Having found the right pattern for a given problem, a developer must then apply it. Examples of patterns include: an analysis pattern for hierarchical relationships between organizations and/or people, a design pattern for maintaining an audit trail, a design pattern for applying different **levels of security** to different user types, and a design pattern for composite relationships between objects.

There are many frameworks within the Java programming environment. For example, Java Security, a very important topic in new netcentric architectures, provides a Java Security Framework. This is a plug and play framework that allows developers the **option of plugging in a security** provider of their choice (DES, RSA, etc) or developing a custom security solution that can be called by security clients. To create a new security provider, the developer must only implement the required interfaces for the framework and provide a well-known name. Once these requirements are met, the component can be plugged into the framework.

The next snippet of code is for the Customer Component (Server). The interface delegates the processing to the component. // Get the Customer's data and return it. public CustomerStructure getCustomer(String aCustomerId) [// Go to the database and get the // customer with the appropriate ID Customer aCustomer= . . . // Create a structure and populate it with the // **customer data retrieved** from the DB. CustomerStructure

aCustomerStructure=new
CustomerStructure( );
aCustomerStructure.id=aCustomer.getId( );
aCustomerStructure.status=aCustomer.getStatus( );
aCustomerStructure.firstName=aCustomer.getFirstName( );
aCustomerStructure.lastName=aCustomer.getLastName( );
return
(aCustomerStructure);] public String getPhone(String
aCustomerId) [ ] public
AddressStructure getAddress(String aCustomerId) [. .]

In an option, the use of the locally addressable interfaces may be facilitated
by structured-based communication. As another **option, the access** may be
allowed via a customer interface proxy, a customer server and a database of the
globally addressable interface.

Collaborations 8. The Client asks the Customer Proxy for the data associated
with the Jimbo Jones object. 9. The Customer Proxy forwards the **request**
across the network to the Customer Component 10. The Jimbo Jones object
creates a data structure and populates it with its data. 11. The Data
Structure is passed across the network to the Customer Proxy on the Client.
12. The **Customer Proxy forwards the data** structure containing Jimbo Jones'
data to the Client component.

The following block of code is the "main" routine for the Client. Even though
all distributed calls are made through a proxy, the proxy code is not shown in
this example. The two proxy classes encapsulate the details associated with
distributed network calls. // Client side code here. Main() [// Create a
Proxy to the Customer Component. CustomerComponentProxy
aCustomerComponentProxy=new CustomerComponentProxy(); // Get a Proxy to the
Jimbo Jones Customer // object. CustomerProxy
aCustomerProxy=aCustomerComponentProxy.getCustomer("Jimbo Jones"); // Get

Customer data from the Customer Server // Component(Call across the network)
CustomerStructure
aCustomerStructure=aCustomerProxy.getCustomerAsStructure();
// Use the Customer data received in the // structure. For Example, display
the data // structure data (aCustomerStructure) in a UI.]

The following code is a sample Customer Server Component. The Customer Server
Component is used to retrieve the data associated with customer Jimbo Jones
from the database. It also instantiates a customer object using the data
retrieved from the database. // Customer Component Code here public class
CustomerComponent [// Put the data associated with a Customer Object // into a
data Structure. This data structure // will be sent across the network to a
client. public Customer getCustomer(String aCustomerName) [// Find the
Customer in the database . . . // Instantiate the Customer Object Customer
aCustomer=new Customer(.. . . // Return a "remote object reference" to the //
Jimbo Jones Customer object. return (aCustomer);]]

The predetermined criteria may include user preferences, an experience level of
a user, security profiles, and/or workflow settings. Also, the activity may be
allowed to run without a corresponding view. The activity may also operate on
a machine separate from a machine of an end user.

The View Configurer then determines which view to launch. This can be based on
a variety of criteria, such as user preferences, experience level, security
profiles, or workflow settings. The configurer determines the correct view and
attaches it to the activity underneath.

Benefits Common User Representation. One single representation of a user and
their access rights can be shared across all areas of the

system. Extensible
Security. Because there is one source for the User Context various policies or
strategies could be used to identity and authenticate the User within a
context. For example, it could encapsulate the User's certificates that allow
more advanced security strategies to determine authorization. Class
UserContext UserContext(Identifier identifier) Identifier getIdentifier()
String getName() void setName(String newName) void addRight(String accessArea,
AccessLevel level) void removeRight(String accessArea, AccessLevel level)
Vector getRights(String accessArea) boolean canCreateIn(String accessArea)
boolean canReadIn(String accessArea) boolean canUpdateIn(String accessArea)
boolean canDeleteIn(String accessArea) Class AccessLevel static AccessLevel
create() static AccessLevel read() static AccessLevel update() static
AccessLevel delete() boolean=(AccessLevel anAccessLevel)

The patterns in this section solve several of the fundamental problems
encountered in the development of an object-to-relational persistence
architecture, including the mapping of classes to tables (Data Handler,
Individual Persistence), identity management (Object Identifiers as Object),
caching(Object Identity Cache), allocation of responsibilities (Data Handler,
Piecemeal Retrieval, Persistent State Separate from Persistent Object), and
data access optimization (Multi-Object Fetch) and the mapping of basic SQL
types to object attributes (Attribute Converter).

Individual Persistence assigns responsibility for data access at the level of
individual domain objects. Each domain object or class can retrieve, update,
insert, and delete its data from a persistent store independently of other

objects or classes. This promotes encapsulation and reuse across business
transactions.

FIG. 163 illustrates a flowchart for a method 16300 for organizing data access
among a plurality of business entities. **Data about a user is retrieved** and
packaged into a cross-functional data object in operation 16302 and 16304. A
request for data is retrieved from one of a plurality of business objects in
operation 16306. In operation 16308, the business object are directed to the
data object such that the business object retrieves the entire data object.
The business object also selects the data from the data object.

Both locking and integrity may use a uniform mechanism. The business object
may **retrieve** account, **customer, and bill-related data** from the data object.
Also, the business objects may be able to update themselves independently of
each other.

Using the same bill payment example, a legacy system might utilize a `accept
bill payment` transaction. This transaction would require only a small portion
of the **attributes for the account, customer,** or monthly bill entities and so
the data would be **retrieved** piecemeal only as required by the transaction.

Once the account **retrieves all of its data, it will know its unique customer**
identifier. The **customer can then retrieve its own data,** which includes the
identifiers for the credit profile and both addresses. Finally, the profile
and addresses can retrieve their data. In this case, profile and address
retrieval depends on **customer data; customer retrieval** in turn depends on
account data.

Although LUW contexts manipulate separate copies of business objects, they can often share the same retrieved data stream. For example, when a workstation **retrieves data for Customer** ABCD, the returned stream can be stored in a global cache. If another context wants to later instantiate its own copy of Customer ABCD, it can reuse the details stored in the stream cache. This improves performance, by avoiding a redundant **request** to the remote data store.